4.0 COE Database Concepts

The function of a Database Server together with the databases it manages is to provide information to all users through applications that access the databases, and to support system and database administrators maintenance functions. The operations of a database server involve the database server itself, the databases managed by it, and applications that access one or more databases. The discussion that follows addresses the operational roles of each.

A COE database server provides data management services to all applications. In order to be useable it must be a stable, reliable operating environment that developers can design to. Database services include tools to support the management, by a site administrator, of users' discretionary access to databases based on the applications they are permitted to use. This is governed by the following principles.

¥ Users will not need access to all applications.

¥ Applications will have multiple levels of database access that can be granted to users.

¥ When access to an application is granted to or revoked from a user, the corresponding database permissions are also granted or revoked.

COE databases are a federation of application-owned and common databases. Application segment developers control the data and structures that are specific to their segments and can change those data or their structure when necessary. The common, corporate databases are owned by DISA; their data and structures are centrally controlled. These databases reside on the Database Server that provides services to the applications, acting as database clients, within the network. The databases within a particular database server are isolated from each other, physically and logically, by being placed in separate storage areas and by being owned by different DBMS accounts.

This configuration, using the disk controller and drive analogy is shown in Figure 4-1. The core database configuration, containing the DBMS Data Dictionary and associated system information is part of the COE and is represented by the System Database. All other databases, whether provided by DISA, a developer, or some other agency, are included as 'component' databases under the management of the Database Server. The set of component databases available from a particular database server is determined by the set of applications that server's database is expected to support.

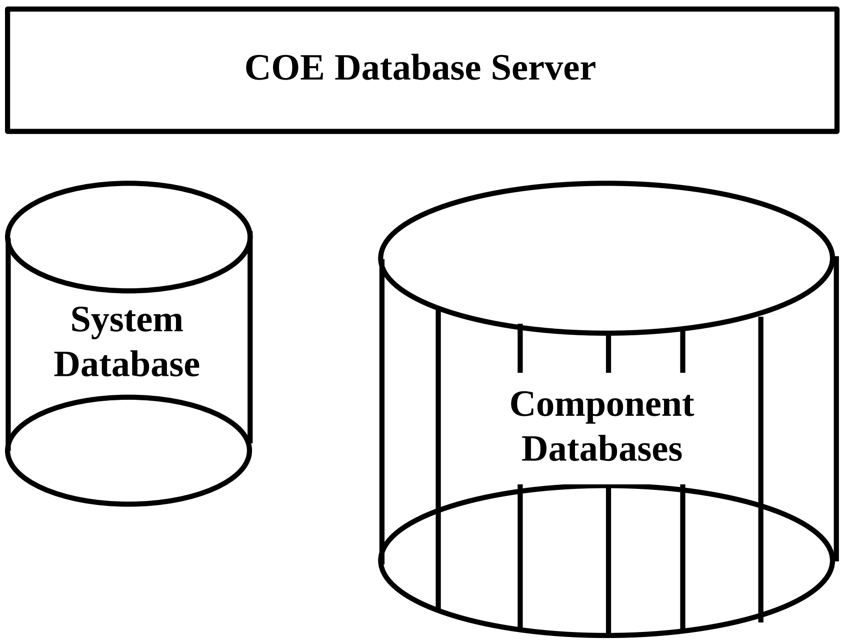

Figure 4-1: Database Server Architecture

All databases are shared assets, whether they are common or not. They are also dynamic because their data can change even while structure remains static. Databases may be interdependent. Databases depend on the COTS DBMS service and are built within its constraints. Databases can be accessed by applications other than those written by the database developer. (While most database applications are usually written by the database's developers, it cannot be assumed that this will be the only case.)

Applications that use databases to manage their information are the interface between users and the data. Some applications use their databases interactively, in a transaction processing mode, to perform the work for which they have been designed. Others have a single process that writes data, and many readers. Some pull data from remote sources to replace existing data the then allow read-only access to that remotely provided information.

Users connect to the database server through the applications, possibly in multiple sessions. Each session must behave as if it is isolated from the rest of the system and knows of no data other than those belonging to the application it is executing.

4.1 Constraints on Database Developers

The developers of databases and applications accessing databases must conform to the strictures of the COE database server environment so they do not bypass or corrupt its features. The combination of the database server configuration and the developers’ implementations must ensure two things. First, each connection of a user to a database through an application must function in the proper context for that application and database. Second, each user’s connection to a database must not interfere with any other user’s connection the same or any other database.

The development and integration standards for COE databases support an evolving configuration of database services. In GCCS version 0, each application has its own database and database management system. Commencing with GCCS Version 1.1, the separate database servers were replaced by a single server running a single instance of the database management system (DBMS). Each application retained ownership of its database within that instance, but shared the DBMS service with the other applications’ databases. Thus, the database server provides shared, concurrent access to multiple databases with varying degrees of autonomy. COE-based systems are to follow the same approach as that pioneered by GCCS.

The single server, single instance data management service conserves system resources by not requiring multiple copies of the DBMS to be executing and eases system management by providing a single point-of-entry for database management services. That single point of entry also simplifies application development. The benefits that come with the central service do limit the freedom of developers by requiring that they implement databases compatibly with the larger multi-database environment. In addition, the increased complexity of a multi-database system could overburden the operational sites’ system and database administrators unless it is implemented consistently. This again limits developers by constraining their databases to function within a consistent administrative framework.

The principal consideration for developers is that their applications and databases no longer have exclusive use of the database management system. Instead of being an application-specific data management tool, the DBMS is a central service that supports all applications’ databases. As a result, developers cannot customize or tune the DBMS to the particular behavior of any application. Any such modifications to the DBMS will inevitably affect other applications and databases. Similarly, the individual component databases are no longer the sole occupants of the DBMS. Developers must implement their applications and component databases so that they do not interfere with others sharing the same DBMS. Further, because there are multiple databases in the DBMS, applications can connect improperly to other databases. Developers must ensure that their applications connect only to the database they are intended to use. They must also design their databases to maintain their own integrity without reference to external applications.

In order for component databases to plug into and play properly on a Multi-Database Server, they must conform to the standards defined herein. The objective is to support the independent development of maintainable databases that will function reliably within the larger multi-database system.

Developers must implement their databases such that the operational sites' administrators can manage the collection of databases. If system and database administrators are required to manage multiple databases, each with its own integrity rules and access methods, their jobs quickly become impossible. This means developers must adhere to common implementation methods.

4.2 Database Integration Requirements

The Database Server is the COE component that provides shared data management within COE-based systems. Regardless of the COTS DBMS used to provide database services, its functions within the system remain the same.

- Support independent, evolutionary implementation of databases and applications accessing databases.
- Manage concurrent access to multiple, independent and autonomous databases.
- Maintain integrity of data stored in the DBMS Server.
- Provide discretionary access to multiple databases.
- Sustain client-server connections independent of the client application's and database server's hosts.
- Support distribution of databases across multiple hosts with replicated data and with distributed updates.
- Provide maintainability of users' access rights and permissions.

In addition, database services within the COE are not restricted to a single vendor's DBMS. As a result, developers must implement their databases such that dependence on any particular DBMS vendor's product is limited. The discussion that follows provides more detail on each of these general requirements.

4.2.1 Evolutionary Implementation

The goal of evolutionary implementation is to be able to incrementally develop, field, and improve software and information services. This "build a little, test a little, field a lot" philosophy applies to databases as well as applications. In the database context, the objective is to field the latest and best information structures and contents. Databases and applications should be able to evolve independently in principle, but in practice they are not likely to because of the dependence of applications on the database's structure. In addition, the component databases are dependent on the DBMS for their implementation.

Database developers can still support evolutionary implementation by maintaining the modularity of their component databases. To achieve this goal, component databases must first coexist within the server without corrupting each other's data. This is not requiring databases to be isolated from each other; it is requiring that all actions across database boundaries be intentional and documented. The COE architecture requires that segments not modify other segments. The same applies to component databases modifying or extending other database. When a database does have a dependency on some other component database, that dependency will be kept in a separate segment.

Component databases are dependent on the DBMS used for the database server. The specific commands used for their implementation within the DBMS, and the environment it provides are both provided by the DBMS vendor. Database developers must be careful in their use of vendor-specific features so they do not create unintended dependencies on specific database management systems or, more importantly, particular versions of the DBMS while still taking advantage of the database server's capabilities. To accomplish this, developers shall separate DBMS-specific code from that which is transportable.

The same constraints on databases also apply to applications accessing those databases. Application developers must ensure that applications connect through regular, documented API’s and shall not assume the use of particular DBMS versions. This does not prohibit developers from using DBMS vendor supplied tools that are part of the COE.

The key to managing the evolution of component databases and the applications that use them is documenting their interrelationships. Applications’ dependencies on databases shall be documented so that database version changes can be tested with the applications. The component database's dependency on the DBMS will also be documented for the same reason. If developers use DBMS vendor supplied tools to implement applications, the dependency on the tools will be documented. When applications or component databases access data objects belonging to other component databases, the dependency among the databases shall be documented as well.

As database federation evolves, it is likely that component databases will be upgraded before applications that access them. When applications are affected by component database modifications, legacy views may be provided as directed by the DISA Chief Engineer. Such views will be read-only, but can allow query tools to continue to function until they are modified to work with the re-engineered database.

4.2.2 Managing Multiple Databases

As stated earlier, the COE database architecture is that of a federation of databases with varying degrees of autonomy. Federated means that the component databases are co-located and share DBMS resources. They process data cooperatively but are not part of an overall schema. In some cases they may also share or exchange data. Autonomous means that each database remains an independent entity. Individual databases may be modified or upgraded without reference to others. They are also responsible for maintaining their own data access and update rules.

The federated architecture provides the same modularity within the Database Server that mission application segmentation does for the user interface. The set of databases available from any particular database server is tailored to the information needs of the individuals using that server. Databases that are not needed can be omitted.

In order for this to work, each component database must be implemented in a self contained manner. This is not to say that a database supporting a set of applications should be self-sufficient. One goal of the modular database implementation is to limit the redundancy of information among component databases. Developers should not incorporate information in component databases that is already available from other, existing databases.

4.2.3 Data Integrity

Data integrity addresses the protection of the information stored within a database management system. There are three general circumstances that must be addressed. First is the prevention of accidental entry of invalid data. Second is the security of the database from malicious use. The third is the protection of the database from hardware and software failures that may corrupt data. The implementation of appropriate data integrity measures is the responsibility of the database developers using the features of the DBMS.

The Database Server is responsible for preserving the integrity of each component database and for preventing connections between an application and data that belong to any other application. COE-based systems are typically secure systems that contain and process classified data. The database management component must conform to the security policies and practices of the overall program. Otherwise, the database server supports the data access restrictions and integrity assumptions incorporated in each database.

The Database Server provides the basic functionality expected of a DBMS. It ensures the recoverability of failed transactions or of a crashed system. The atomicity, concurrency, isolation, and durability of database transactions are the responsibility of the applications accessing the server. However, supporting these transaction properties is the server’s responsibility. Developers must pay special attention to transaction isolation because of the multi-database configuration of most COE-based systems.

Database developers are responsible for defining and implementing the integrity constraints of their databases. The Database Server is responsible for enforcing the developers integrity constraints when they are defined within the database. Application developers must ensure that their applications connect properly to their databases and do not connect improperly to anyone else’s databases. Adoption of these practices protects all applications’ data and allows the database server to maintain all databases reliably.

Within a component database implementation data integrity takes the form of what are often called constraints and business rules. In the current context, constraints are defined as the rules within the database that govern what values may exist in an object. Business rules are those rules within the database that govern how data is updated and what actions are permitted to users.

There are two flavors of constraints. The first type of constraint is structural; it concerns such things as the uniqueness of a primary key field or the equality of foreign key fields between records in different data objects. The second type of integrity constraint addresses the values permitted in data fields. It may restrict the values in a field to a certain numeric range or a list of permitted key words. It may assert relationships (e.g. arithmetic) among multiple fields.

Business rules are also constraints, but have more to do with the information than the data object. A simple rule might require that an entry be written to an audit table each time a record in some table is inserted, updated, or deleted. Another rule might require information to be transferred from one site's database to one or more other sites when certain combinations of actions take place or after a specified time period.

Until recently, commercial database management systems were limited in their ability to support the variety of constraints and business rules that may be needed in a database. As a result, most of the constraints and business rules of databases have been implemented in the applications, not the database. Because of the federated database architecture and because the applications that maintain those databases are developed independently, it is difficult to ensure uniform and consistent enforcement of those rules and constraints.

Developers shall place their business rules and constraints in their databases rather than their applications. This keeps control of data maintenance access in the hands of the developers where it belongs and ensures that constraints cannot be bypassed. Developers have the knowledge of their constraints and business rules; DBA’s and users do not.

The reason for placing constraints in the database is shown in Figure 4-2. Application One and its associated component database were implemented with business rules and constraints in the application. Application Two placed those constraints in the database. When a third application accesses both databases, it is unaware of Database One's business rules because they are inaccessible. If this application, which could be a user-developed query tool, modifies Database One, it could corrupt the database out of ignorance.

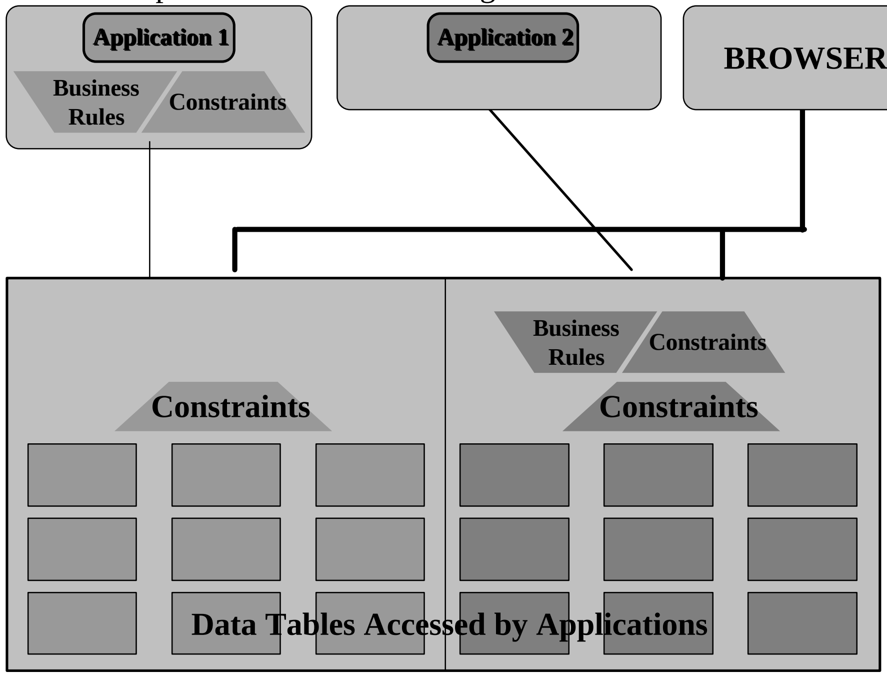

Figure 4-2: Business Rules and Constraints

Placing business rules and constraints in the database promotes client-server independence. The efficient implementation of constraints and business rules will have to make use of the DBMS's capabilities. If these rules are in the component database, the application is less dependent on the COTS product. Also, this approach reduces network communications loading by allowing the DBMS, rather than the application, to enforce the rules. Checking rules within the database avoids passing multiple queries and their results over the network between the DBMS and the application.

4.2.4 Discretionary Access

Discretionary access addresses the selective connection of users to databases through applications. Database access is discretionary because not all users have the same permissions to use applications. The objective is to ensure that users' database connections operate in the proper context for the applications. Users must be able to operate several different applications at the same time. The DBMS server must effect each application’s accesses to different sets of data objects. This means permission to access to specific tables and the mode (read or write) of that access. Because several databases may exist on the Database Server, each application must be written to access only the database it belongs with; it must be unable to access tables belonging to some other application. Each user-application connection have only the permissions needed for that context.

There are three components to this issue: Session Management, Discretionary Access Control, and Access Management. The first refers to the DBMS' ability to keep different connections separate. The second addresses the context of an individual connection. The third deals with the requirements of system and database administrators to manage the accesses that are provided to users. Without the correct functioning of all three components, data integrity and consistency can be compromised.

In order for this system to be useable, it must provide support to systems administrators as they manage users' discretionary access to subsets of applications and databases.

4.2.4.1 Session Management

A session is an individual connection between an application and the database management system. It is the means by which the database server isolates one user's activities working with an application from all other users that are connected to the DBMS. In this context autonomous applications such as message parsers are also database users.

The Database Server is responsible for session management as shown in Figure 4-3. In this example, two users are connected to the database server. The first has two sessions with application A and one with application B. The second has a session with application B and one with application C. The database server maintains five separate sessions. Two sessions are connected to component database A, two to component database B, and one to component database C; no session is connected to component database D.

Note that each different execution of an application is considered a separate session and is functionally isolated from other executions of the same application. Thus, when User 1 starts two separate instances of application A, the DBMS treats them as different sessions (A1 and A2). This ensures that changes being made in different sessions propagate correctly and do not corrupt data accessed by other sessions.

The key points with respect to session management are that the DBMS, in managing connections, provides sessions and isolates each one from all others. Isolation sustains transaction management and system recovery. It also supports the traceability of database transactions to the user and application.

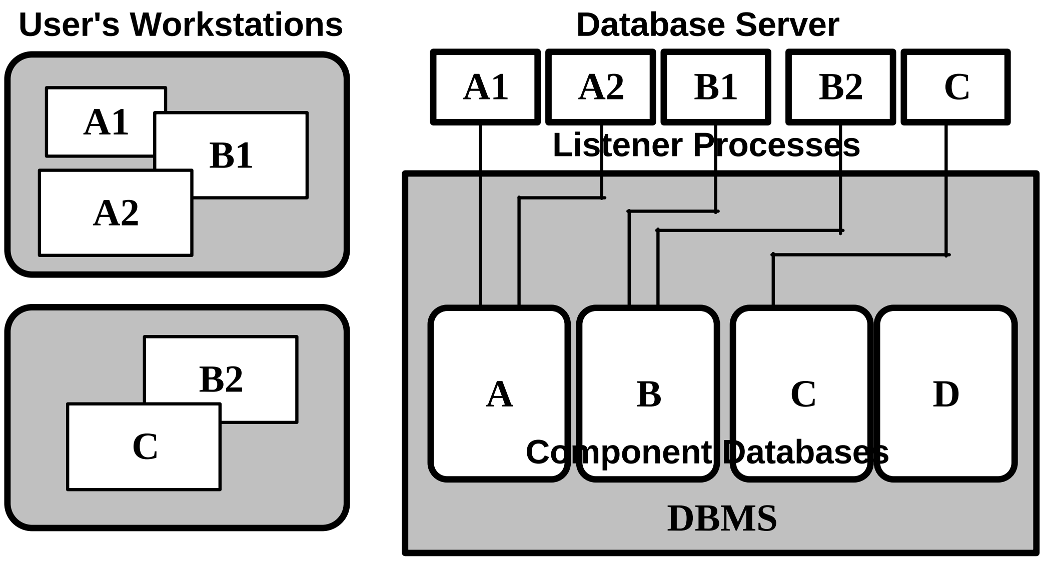

Figure 4-3: Session Management

4.2.4.2 Discretionary Access Control

Discretionary access control is used to manage users' permissions to employ applications to access or modify data managed by the database server. It has a broader scope than information security. Security is focused on whether users are permitted to know about and allowed to view certain information. Discretionary control of access deals not only with users' permissions to change information but also the context in which they are permitted to make changes. Users will have access to multiple databases through many different applications. Their overall database permissions are the union of the permission sets of the individual applications they have the right to use.

Figure 4-4 illustrates the need for discretionary access. A user has three database sessions active, one with each of three different applications. Each application accesses a different set of objects within the database. The data objects shown represent all objects that user has permission to access and are marked to show which application context is relevant to that access. If all of the user's database permissions were active at all times, it would be possible for one or another of the applications to access and modify data that is not relevant to it. Instead, each application must only be able to access its corresponding data objects.

It is the responsibility of database and application developers to provide discretionary access controls. The operational sites' administrators are responsible for using those controls when assigning database and application privileges to users. Session management by the DBMS provides the database and application developer the isolation needed to implement discretionary access. When designing access controls the following principles apply:

- Users shall have unique accounts within the DBMS. Those accounts shall have only the database permissions needed for their work

- A user's database permissions will only be active within the context of the current application and database session. In other words, when a user starts a database session through some application, that session will only be able to access the data objects appropriate to the application and the only active permissions on those objects will be those appropriate to that application's use of those objects.

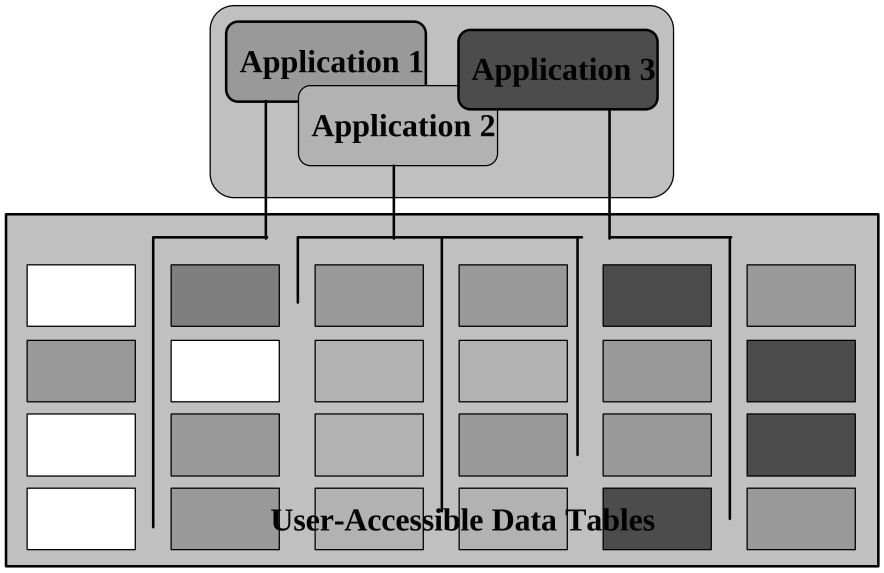

Figure 4-4: Functional Context

These context specific controls are necessary because users will have access to multiple applications and each application has its own set of database permissions. As a user is granted access to data objects based on the applications that he/she needs to use, the total set of database grants for that user expands. The DBMS manages sessions at the user account level, so each user has all granted permissions on all objects whether they are relevant to the current session or not. If access were not dependent on context, users could have inappropriate permissions for a particular session. For example, a user might be able to write to a data table that is supposed only to be read by the current application. Such pathological connections to data objects will, almost inevitably lead to data corruption.

The context in which an application operates on the database is the application's "database role." A database role is the minimal set of database permissions needed for an application to function correctly. Since these roles are linked to the application, their definition is the responsibility of the application developer. However, the roles are implemented within the database so they become part of the database segment.

4.2.4.3 Access Management

Access management addresses the work of system and database administrators giving users the permissions they need. They must be able to connect users to applications and to databases. They must also be able to revoke or modify those connections as users transfer or assume different responsibilities. The large number of applications and databases available within COE-based systems could make the administrators' tasks unmanageable if access management is not supported with the proper tools. This section discusses the developers' responsibilities for supporting access management.

The act of adding an application to a user’s access list, menu, or whatever, entails the adding of associated database permissions to that user’s DBMS account. Similarly, revoking access to an application requires that corresponding privileges be revoked within the DBMS. Users must have the proper permissions on both the application and the database, so the two system activities are interdependent. Access to applications will often be granted in logical sets or groups of related applications. As discussed in the previous section, access to databases must be linked to each individual application or the functional context of the application is lost.

The grant association process is illustrated in Figure 4-5. A user is being given permission to use three applications. As a part of that process, the user must also be assigned the database roles associated with those applications. Through the database roles, the user receives the permissions on the data objects needed to use the applications. If, later, the user no longer needs to use these applications, the administrators can reverse the process. When the application permission is revoked, the database roles are withdrawn from the user. The other reason for managing database roles at the application level can also be seen here. Assume that these applications represent a group that are accessed together and that have identical database permissions. If the grouping of applications changes at some point, the collective role might not be valid. In addition, if there is not a one-to-one correspondence between applications and database roles, it becomes impossible to determine when a database role should be revoked.

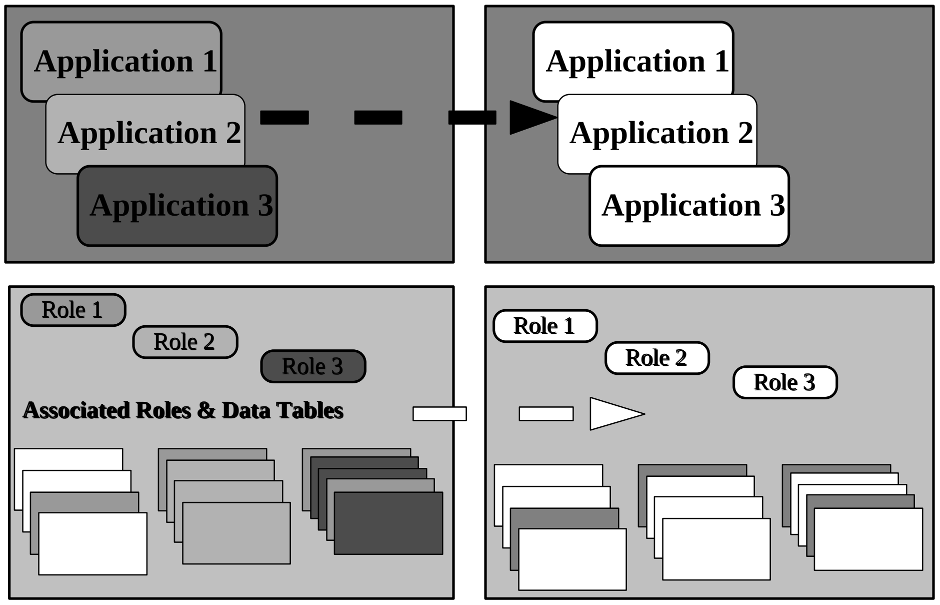

Figure 4-5: Grant Association Process

Database application developers are the only ones with comprehensive knowledge of interactions with the database. They must define the database roles and provide the scripts or command sets that create them for inclusion in the database segment. The scripts that grant and revoke database roles are part of the application segment. This allows them to be executed by the system's administrators when they are managing access to the applications.

4.2.5 Supporting Multi-Database Tools

Access to multiple database is one of the major benefits that the COE brings to its users. Database browser tools, such as APPLIX, allow users to construct ad hoc queries that span different subject areas and that are not supported by mission-specific applications. At the same time, however, such multi-database applications present special problems in the COE context. If the databases were read-only, browsers would not cause problems. However, many databases are designed to be maintained interactively using the applications associated with them. This means that users will have permission to write to databases. Those write permissions are potentially active when a user is using a database browser which means that the browser tool can also write to COE databases. Since the browser is independent of the applications designed for particular databases, it will be ignorant of any constraints or business rules that are in those applications. Thus it could corrupt data due to its ignorance of the rules.

The key to ensuring database integrity in this case (as in all others) is the enforcement of constraints and business rules within the database. If the rules are part of the database, they cannot be bypassed.

The second issue is one of understanding the context of a particular database. When users formulate queries that span multiple databases, they are likely to encounter differences in the way information is represented among those databases. This could lead the users to draw erroneous conclusions from their query results because they do not understand the differences between the databases. To limit the chances of this, component database developers shall provide comprehensive information on their databases to be incorporated in the DBMS data dictionary. This information is part of the database segment.

4.2.6 Client-Server Independence

The COE uses a client-server architecture. This applies to database services as well. Developers must preserve the independence of their applications, functioning as DBMS clients, from the database server. Specifically, applications that access databases must not be built so that they have to reside on the database server in order to work correctly. It cannot be assumed that all operational sites will have a local database server. Further, where sites have a local database server it may be on a separate machine hosting that is dedicated to the DBMS, or the server may be co-located with the application on a single machine acting as the application server and the database server. To maintain independence and support the client-server architecture, applications cannot assume they reside on the database server.

To sustain the independence between DBMS clients and the database server, developers must not mix extensions to the DBMS with their databases and must separate the database from the applications that use it. If specialized data management services are needed by particular applications and are not part of the COE's database services, the provision of such services must be approved by and coordinated with the DISA Chief Engineer.

For example, assume some application needs a COTS expert system shell to manage a knowledge base that is a component of the application and that it interacts with the database server. The expert system shell, to work properly, has to be co-located with the DBMS. The expert system then becomes a segment that is separate from the application that uses it.

4.2.7 Distributed Databases

A distributed database is one whose data are spread across multiple sites. Data are replicated in a distributed database when copies of particular objects or records exist in more than one of those sites. Data are fragmented when they are split among sites. Databases are distributed (fragmented or not) to improve responsiveness and increase availability in systems that serve geographically dispersed communities. Databases are replicated to enhance their survivability in the same circumstances. In either case, one implementation objective of any distributed database is to provide location transparency. This means that the user need not know where data are located to be able to access or work on them.

Depending on the component database, the COE supports several flavors of distributed databases. Some current databases are relatively static and are replicated at multiple sites, but exist independently. They are updated through the periodic replacement of information at each site that has a copy. Others, such as the JOPES Core Database, are dynamic and are replicated concurrently across several sites for survivability. They use transactions to effect updates across the affected sites.

The COE provides distributed database management services for the developers of distributed databases so they can maintain location transparency and distributed transaction processing. The specific services implemented for a particular COE database system will depend upon the nature of its distributed data. GCCS, for example, uses an asynchronous transaction model. A financial system may require the use of the more restrictive, but synchronous, two-phase commit.

The technology that supports the distribution of databases as used in the DII COE is evolving rapidly. Accordingly, the GCCS program does not, at present, prescribe a particular implementation method. Developers of distributed databases must coordinate their activities with the DISA Chief Engineer to ensure that their approach can be supported and is consistent with the objectives of the broader program. When a distributed database is implemented, developers should keep in mind that the distribution plan (fragmentation schema) may change over time. Distribution methods and the tools used to support them will also evolve as technology matures. Where developers are assigned responsibility for database fragmentation schemas, each fragment shall be in a separate segment so different schemas can be implemented.

The distribution of data also means that users potentially have access to multiple database servers. The assignment of users to servers will depend on the distribution schema as implemented for the various sites. The Sites’ DBAs are responsible for aiming users’ processes at the correct database server. Developers shall not assume that users are attached to a particular server. Developers’ applications shall not modify the user’s DBMS environment to associate them with a particular database server.

4.3 Guidelines for Creating Database Objects

This section provides guidelines for developers in creating their database segments. Its objective is to support consistency across different databases and improve the mutual independence of the database federation.

4.3.1 Database Accounts

Three categories of database accounts have been defined within the COE: DBA's, Owners, and Users. They have different functions and levels of access to the DBMS based on those functions.

4.3.1.1 Database Administrators

The Database Administrator (DBA) accounts have access to all parts of the DBMS. They are to be used only for system administration. Their use by database server segments is prohibited except during the installation process as discussed above.

4.3.1.2 Database Owners

These accounts are the creators and owners of the data objects that make up an application server segment. The name must be unique within the COE community. Developers will normally use the segment name or a variant of it as the owner account name. Owners accounts must have their password changed after a database installation. Users shall not use the owner accounts to connect to databases.

4.3.1.3 Users

User accounts belong to the individuals accessing databases Each individual must have his/her own unique user account. Creation and maintenance of user accounts is a site DBA responsibility. Developers shall not assume the existence of particular users and shall not create user accounts.

4.3.2 Physical Storage

Database management systems provide file management transparency across multiple host computer systems by hiding the details of file storage from the database's data objects. At the same time, however, the placement of data objects on physical storage devices has an impact on system and database performance due to disk contention and other file system access issues.

4.3.2.1 Data Store/File Standards

Data can usually be grouped into logical sets based on the source, type of information or use of the data. These logical sets of data are defined as a data store. An application's data will normally be a single logical set and hence one data store. Its name will be an identifier for the data store. GSORTS is an example data store name.

The data store identifier will be used as the database schema name to clearly identify the logical set of information. This same identifier will also be the owner account name.

Developers will provide the scripts to create their DBMS storage and the operating system files associated with them. The files will be created in the `DBS_files` subdirectory of the application server segment's directory tree.

4.3.2.2 Data Storage Standards

Developers should define one or more storage areas for their database segments. The objective is to allow DBAs to place data files on separate physical storage devices based on the tablespace's use within the DBMS. Storage area names will be meaningful and a maximum of 30 characters (letters, numbers, and underscores). The name is not case sensitive. No DBMS reserved words shall be used.

Storage area names must also be associated with the segment and data store identifier. Most applications will have either two or three storage areas: Data, Indexes, and (if needed) Static data. The following naming convention is to be used: <*segment prefix*>_DATA, <*segment prefix*>_INDEX, and <*segment prefix*>_STATIC for the three storage areas respectively.

One or more data files may be created to support each storage area. The data file names should be chosen so they are clearly associated with the tablespace. The recommended naming convention is <*segment prefix*>_<*store type*><*n*>.dbf where 'store type' is the storage area's purpose (e.g. index) and 'n' is a one-up serial number for the file. An example would be `gsorts_data1.dbf`.

4.3.3 Database Objects

The definition of a database schema – the set of data objects, their interrelationships, constraints, and rules for access or update – is the responsibility of the developers. Developers shall not duplicate data objects that are part of the corporate databases provided by DISA. Where possible and appropriate, developers shall take advantage of and share objects belonging to other databases within GCCS. To this end, developers shall provide the definitions of their schema components for inclusion in the DBMS data dictionary.

Developers shall provide DISA with their proposed database schema early in the segment design process. The schema will be reviewed for duplication of objects in other component databases.

4.3.3.1 Database Tables

Table names will be meaningful and a maximum of 30 characters. If Oracle database snapshots are being used for data replication services for other sites, developers should limit the table name to 24 characters. Oracle uses six characters to identify a database snapshot.

4.3.3.2 Data Elements

Data element names will comply with DISA/JIEO or DIA standards where applicable. Within a schema developers should use the same data type, size, etc., for all occurrences of the same name. If elements are chosen from Joint standards, they shall use the data type and units of measure prescribed in the standard.

Developers shall not use data types that are machine dependent. This applies primarily to numeric data. The value ranges of the ‘float’, ‘double’, and ‘real’ data types are machine dependent in both Oracle and Sybase.

4.3.3.3 Database Views

Views are often used to restrict users’ access to vertical or horizontal subsets of data tables. Current DBMS’s are limited in their ability to support updates through views. If developers need updateable views, the DBMS’s restrictions must be kept in mind. If the updateable views are required for security or data privacy, developers should not grant users access to the base tables, only to the views.

In general, the following restrictions apply to updateable views.

Horizontal (row-wise) views. Database Servers can support inserts, updates, and deletes through horizontal views. Such views include those where one table is used to constrain the view to a subset of rows in another table. Developers are responsible for implementing appropriate error handling if users try to insert a row that duplicates a hidden row.

Vertical (columnar) views. Database Servers can support updates and deletes through vertical views as long as the database constraints do not reference hidden columns. Inserts can only be supported if all hidden columns are allowed to be null or if triggers are provided to populate them with default values. Developers are responsible for implementing appropriate error handling if a user's update violates a constraint on a hidden column.

Multi-table views. At present, the Database Servers implemented in the COE cannot consistently support data modifications through views of more than one table. Developers should implement such updateable views in applications. These views should be accompanied by comparable read-only views of the individual tables.

4.3.3.4 Rules on Database Objects

Rules on database objects incorporate several different concepts. Their underlying purpose is to maintain database integrity through the enforcement of the constraints and business rules of the database.

For purposes of this document, the following definitions apply. Constraints are restrictions on data elements with respect to the values they may contain. For example, a country code field could be constrained to the set of DIA prescribed two-character country codes. Business Rules are restrictions that occur in the context of database operations that affect multiple interrelated objects and elements or that are beyond the ability of a constraint to express them. For example, any update to a facilities table may require that an entry be written to an audit table recording the ID of the user making the change and the time at which it was made.

Within the Database Server, developers may use DBMS constraints, stored procedures, or triggers to implement either constraints or business rules.

4.3.3.4.1 Constraints

Developers should define all referential integrity constraints (Primary and Foreign Keys) that apply to their database schemas. The information in these constraints is vital for maintaining database integrity. Domain Keys (e.g. the SQL Check constraint) should be used to maintain the validity of column values. Unique columns should be constrained rather than indexed. Constraints should be explicitly named. The names should be meaningful. The recommended naming convention is <*table name*>_<*cons*> where 'table name' is the name or abbreviated name of the table or table and columns involved in the constraint and 'cons' is PK for a Primary Key, FK for a Foreign Key, or CK for a Check constraint.

In most cases developers will wish to create their constraints after the data fill has been completed. The implicit index that accompanies a Primary Key or Unique constraint will slow the data fill significantly. Constraints should not normally reference data objects that are outside the database segment. See below for methods to implement inter-database constraints when they are needed.

4.3.3.4.2 Stored Procedures

Stored procedures are used to maintain database integrity or to enforce business rules when the constraints imposed are too complex for simple SQL constraints. Procedure names should incorporate their schema's name and some meaningful indication of their functions.

Stored procedures should not normally reference data objects that are outside the database segment. See below for methods to implement inter-database stored procedures when they are needed.

4.3.3.4.3 Triggers

Most triggers will be used to maintain database integrity. Others may be used to signal or send data to other, inter-related or dependent database segments. Trigger names should incorporate their schema's name, the trigger type, and some meaningful indication of their functions.

Triggers should not normally reference data objects that are outside the database segment. Database segments should not install triggers on data objects outside the segment. See below for methods to implement inter-database triggers when they are needed.

4.3.3.5 Indexes

Index names will be meaningful without using reserved words. It is recommended that the index name incorporate a reference to the table and column for clarity. Small tables should not be indexed. Indexes should not be used in place of Primary Keys or Uniqueness constraints.

4.3.4 Database Roles

Database roles, in the general sense, simplify the management of user privileges within the DBMS. They are created by the developer to define sets of access privileges that can be given to users by their sites' DBA's. Their names will be meaningful and will be chosen to associate with the segment name. Developers shall provide roles with privileges specific to the applications accessing the database. Each role shall have only the privileges needed by the application it supports.

4.3.5 Grants

Grants allow the DBA to administer and the DBMS to enforce the discretionary access controls required. Developers should grant only the minimum set of permissions needed for the applications that access their databases. Grants should be made to roles/groups and not to individual users.

Granting data access to DBMS ‘PUBLIC’ users is prohibited. Granting data access privileges to user accounts with the ‘GRANT OPTION’ or granting administration privileges on database roles are prohibited.

4.3.6 Inter-Database Dependencies

Where inter-database dependencies are needed they will be implemented as database segments that modify the segment owning the object that creates the dependency. The `Requires` descriptor must identify dependencies on all database segments.

4.3.6.1 Data Objects

Names of objects created in other schemas must identify the inter-database linkage. Developers are responsible for ensuring that their object’s names do not conflict with those already in the schema.

4.3.6.2 Rules in Other Databases

Names of rules created on other schemas must identify the inter-database linkage as well as the rule’s function. Developers are responsible for ensuring that their rule names do not conflict with those already in the schema.